MULTI - LEVEL INTERRUPT SYSTEMS

EE3232 DIGITAL SYSTEMS III
CLASS NOTES CHAPTER 9

Department of Electrical Engineering
University of New Brunswick

 January 5, 2000

SUMMARY

- Objective.
- The need, analogies.
- Interrupt classification.
- 80C188XL interrupts and interrupt instructions.
- Interrupt sequence for software interrupts.
- Interrupt sequence for hardware interrupts.
- Interrupt hardware.
- INA bus cycles.
- Program example.
- Interrupt prioritization.
- Interrupt prioritization using polling.
- Priority encoders.
- Interrupt prioritization using interrupt controllers.
- A simple interrupt controller.
- The 8259A interrupt controller.
- Interfacing the 8259A to a system bus.
- Internal components of the 82C59A.
- Operation of the 82C59A priority resolver.
- Programming the 82C59A.
- Software for interrupt driven applications.
- Example.
- Considerations for using interrupts.
- Reentrant procedures.
- Critical sections.

OBJECTIVE

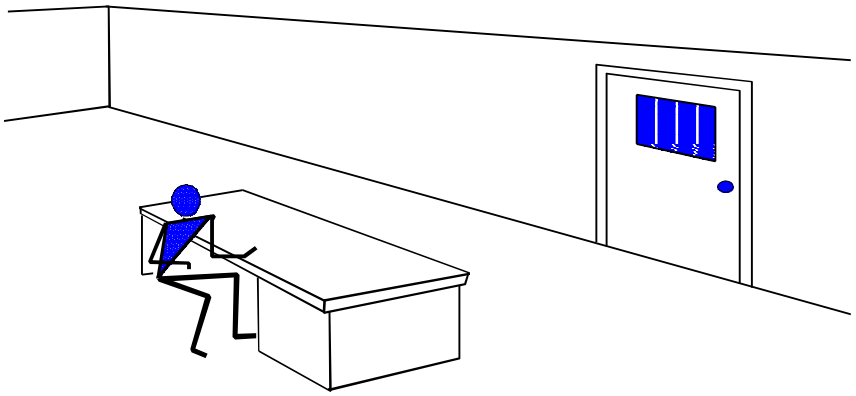
- To concisely describe the sequence of events that occur when a microprocessor responds to an interrupt,
  - In words,
  - Using a timing diagram.
- To design the H/W and S/W interfaces for resolving the priority of multiple interrupts using,
  - Polling,
  - A ***simple*** interrupt controller and
  - 82C59A type interrupt controllers.
- To describe the interaction among the components of each of the interfaces (above) as the priority is resolved.

THE NEED

- ***Interrupts*** - cause a branch of control to an interrupt handler (analogous to a H/W initiated procedure call).
- Used to eliminate ***busy waiting*** associated with synchronizing CPU ← IO and CPU → IO data transfers.
- An interrupt is a form of handshaking, (flip flop with distinct set and clear operations).
- ***The Need*** - throughput efficiency and prioritized service, e.g., power failure, data acquisition systems, disk operations, communication, control applications ...
- Interrupts allows the current program to be suspended while a higher priority task is executed.
- Basis for ***real time*** and ***multi-tasking operating systems***.

ANALOGIES

- Response to an alarm.
- Simultaneous alarms.
- Alarm while responding to a previous alarm.



INTERRUPT CLASSIFICATION

- Interrupts may be classified as,

1. ***Software Interrupts*** (or exceptions) : initiated by an instruction or the result of instruction operation.

int n	Software interrupt of type n.
into	Interrupt on overflow.
÷ 0	Divide by zero interrupt.
trap	Single step interrupt.

2. ***Hardware Interrupts*** (internal and external) : initiated by an internal or external IO device.

   - Non-maskable interrupt input, NMI.
   - Maskable interrupt, INTi.
   - IO device activates NMI or INTi.
   - Hardware interrupts are ***asynchronous*** inputs to the CPU (w.r.t. the CPU clock).
   - Level or edge triggering.

80C188XL INTERRUPTS

Name	Class	Mask	Trigger	Priority	Ack	Type
NMI	H/W	None	(+) Edge & Hi-level	231114	None	2
INTi	H/W	IF	Hi-level		/INTAi	N
int N	S/W	None	None		None	N
into	S/W	OF	None		None	4
÷ 0	S/W	None	None		None	0
Trap	S/W	TF	None		None	1

80C188 INTERRUPT INSTRUCTIONS

IRET	Return from interrupt handler.
STI	Set IF. In “C”, enable().
CLI	Clear IF. In “C”, disable().
HLT	Halt execution until an interrupt occurs.
INT N	Software interrupt.
INTO	Interrupt on overflow.

INTERRUPT SEQUENCE FOR SOFTWARE INTERRUPTS

Consider the instructions, ***int N*** and ***iret***.

```
ENTRY: Initialize interrupt vector table entries for type N.
       Initialize SS, SP, DS registers.
         :
       int N              ; Push flags, disable further
                          ; interrupts :  (IF) = (TF) = 0.
                          ; Push return address.
                          ; Load (CS:IP) with addr of ISR.
                          ;
       Next instruction   ; The address of this instruction
         :                ; is the return address.
       Graceful Exit      ; End of main program.
                          ;
ISR:   Save status.       ; The interrupt handler (also called
       Service IO.        ; interrupt service routine)
       Restore status.    ;
       iret               ; Pop return addr to IP and CS.
                          ; Pop flags.
```

INTERRUPT SEQUENCE FOR HARDWARE INTERRUPTS

1. An IO device activates the interrupt pin of the CPU. (***INTR ← Asserted Hi***). Assume that interrupts have been enabled (***sti or enable() instruction is executed before the interrupt***)* and unmasked.

2. CPU completes execution of the current instruction, then sends an acknowledge to the interrupting device. (/***INTA ← Asserted Lo***, ***starting an INA bus cycle, INTR cleared***)*.

3. The interrupting device sends unique ID info to the CPU which specifies directly or indirectly the address of the interrupt handler. (***The 80188 reads the interrupt type, N, from the data bus as the INA bus cycle finishes***)*.

4. The CPU state and return address are saved. (***PSW, IP and CS are automatically pushed onto the stack***)*.

5. The CPU disables further interrupts. (***IF and TF are cleared***)*.

6. Using the information supplied in Step 3, the CPU branches control to the interrupt handler. (***IP and CS are loaded with the words at (physical) memory location 4*n and 4*n+2***)*.

7. The interrupt handler (interrupt service routine) executes,

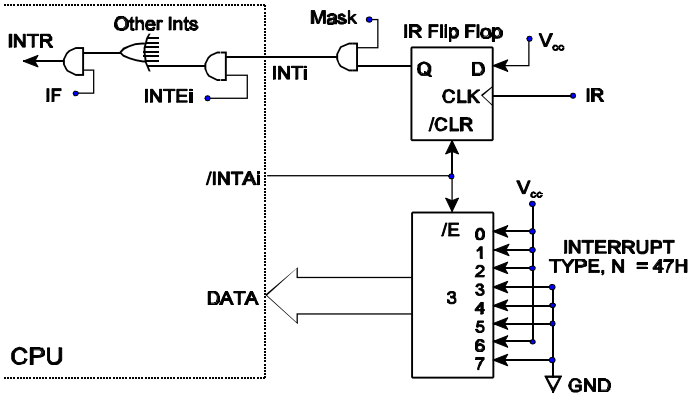
   - The CPU registers used by the handler are pushed onto the stack.
   - The interrupting device is serviced.
   - The CPU registers are restored.
   - The **iret** instruction executes causing :

     Interrupts to be re-enabled.
     Branch of control to the return address.

*** Denotes implementation details of the 80C188XL**.

INTERRUPT HARDWARE FOR A SINGLE INTERRUPT

- Minimal hardware interface for a ***single*** interrupt request.
- H/W consists of an interrupt request flip flop with Mask-bit and a three state buffer for gating the interrupt type.



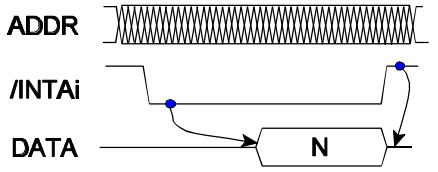
- Be able to explain the need for each component ?

  - The interrupt request flip flop :
  - The three state buffer :
  - The Mask bit :
  - The INTEi bit :
  - The IF bit :

- What if there are multiple interrupt requests with different types ?

THE INA BUS CYCLE

- Analogous to a read bus cycle.
- The CPU reads N over the data bus.



PROGRAMMING EXAMPLE

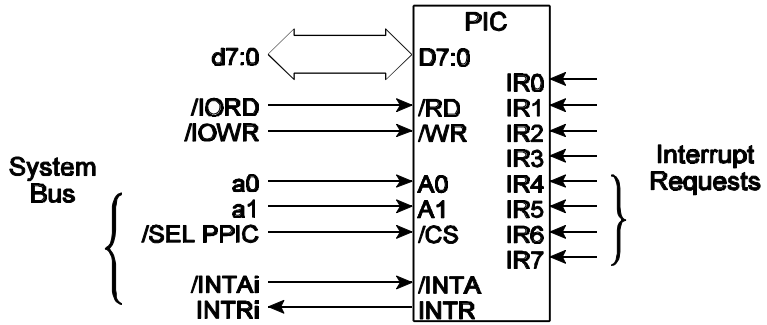
```
_TEXT segment ; ASSEMBLY CODE EXAMPLE                          ; // “C” CODE EXAMPLE
;-------------------------------------------------------------;
; Main Program - Initializations                               ;  void main(void)
;-------------------------------------------------------------;  {
ENTRY:    cli                   ; Disable interrupts.          ;  disable();
          mov N, 23             ; Initialize interrupt         ;  int_vector_init(23);
          call int_vect_init    ; vector table entry at        ;
                                ; location INT_TYPE*4 to       ;
                                ; SEG:EFF address of ISR.      ;
                                ;                              ;
          mov ax, _DATA         ; Initialize DS, SS and SP     ;
          mov ds, ax            ; registers.                   ;
          mov ss, ax            ;                              ;
          mov sp, offset TOS    ;                              ;
;-------------------------------------------------------------;
; Main Program - Operation                                     ;
;-------------------------------------------------------------;
          sti                   ; Set IF to enable ints.       ;  enable();
          mov al, 1             ; Unmask int request at        ;  outportb(MaskP, 1);
          mov dx, MaskP         ; the source.                  ;
          out dx, al            ;                              ;
           :                    ; This code may now be         ;    :
           :                    ; interrupted!!                ;    :
M1:       mov al, 0             ;                              ;
          mov dx, MaskP         ; Mask int off at source.      ;  outportb(MaskP, 0);
          out dx, al            ;                              ;
          cli                   ; IF = 0 disables all ints.    ;  disable();
          int 21H               ; Terminate main program.      ;  }
;-------------------------------------------------------------;
; Interrupt Handler                                            ;  void interrupt ISR(void)
;-------------------------------------------------------------;  {
ISR:      pushf                 ; Save CPU status.             ;
          push ax               ;                              ;
          call service          ; Service interrupting         ;  service();
          pop ax                ; device.                      ;
          popf                  ; Restore CPU status.          ;
          iret                  ;                              ;  }
;-------------------------------------------------------------;
; Initialize Entry in Interrupt Vector Table                   ;
;-------------------------------------------------------------;
int_vect_init:                                                 ; void int_vect_init(void) {
; Initialize interrupt vector table entry                      ;
; for an interrupt type = I_YPE.                               ; asm {
                                                               ;
          push ax                                              ;    push ax
          push bx                                              ;    push bx
          push ds                                              ;    push ds
          mov ax, 0                                            ;    mov ax, 0
          mov ds, ax                                           ;    mov ds, ax
          mov bx, I_TYPE*4                                     ;    mov bx, I_TYPE*4
          mov word ptr[bx], offset ISR                         ;    mov word ptr[bx],offset ISR
          mov word ptr[bx+2], seg isr_0                        ;    mov word ptr[bx+2], seg ISR
          pop ds                                               ;    pop ds
          pop bx                                               ;    pop bx
          pop ax                                               ;    pop ax
                                                               ;    }
          iret                                                 ; }
;-------------------------------------------------------------;
_TEXT     ends
```

INTERRUPT LATENCY

- ***Interrupt Latency*** : is the time between a valid interrupt request (from an IO device) and the time the request is recognized by the CPU.

INTERRUPT PRIORITIZATION

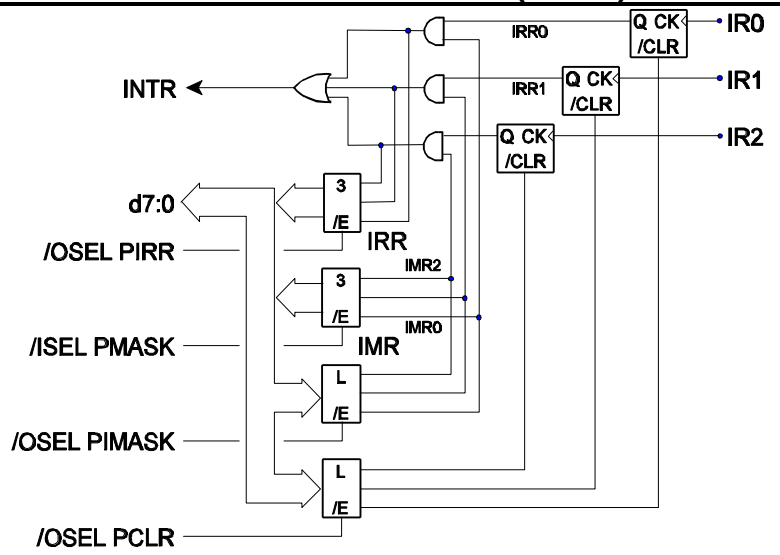
- The CPU responds to one interrupt at a time.
- If two or more interrupts occur simultaneously, then a priority resolving structure in H/W and/or S/W is required.
- Priority may be resolved in S/W using polling or in H/W using a **p**rogrammable **i**nterrupt **c**ontroller or PIC.



INTERRUPT PRIORITIZATION USING POLLING

- Minimal H/W requirements,
  - Interrupt request flips flops with separate clears,
  - Interrupt mask register,
  - Type number interface.
- After a valid interrupt, the CPU polls the interrupt request lines, IRRi , to determine what devices require service.
- Polling takes place within the interrupt handler.
- The software implements a decision to service the unmasked request with the highest priority active IRRi .
- As soon as the decision is made, the associated interrupt request flip flop is cleared.

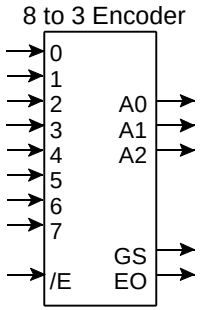
INTERRUPT PRIORITIZATION USING POLLING (CONT'D)




```
void interrupt ISR(void)                    // ISR:
{                                           //     pusha
IRRstate = inportb(PIRR);                   //
if ((IRRstate & 0x01) != 0) { service_0() ; }  //
if ((IRRstate & 0x02) != 0) { service_1() ; }  //
if ((IRRstate & 0x04) != 0) { service_2() ; }  //
   :                                        //
if ((IRRstate & 0x80) != 0) { service_7() ; }  //
                                            //     popa
}                                           //     iret
                                            //
void service_i (void)                       // service_i :
{                                           //
// Service interrupt, i.e., perform IO then       //
// Clear interrupt request flip flop i.     //
                                            //
outport(PCLR, 0 in bit position i) ;        //
outport(PCLR, 0xFF) ;                       //
}                                           //     ret
```

PRIORITY ENCODERS

- A priority encoder provides a unique set of outputs for the active input with highest priority. An example of an 8 to 3 priority encoder.



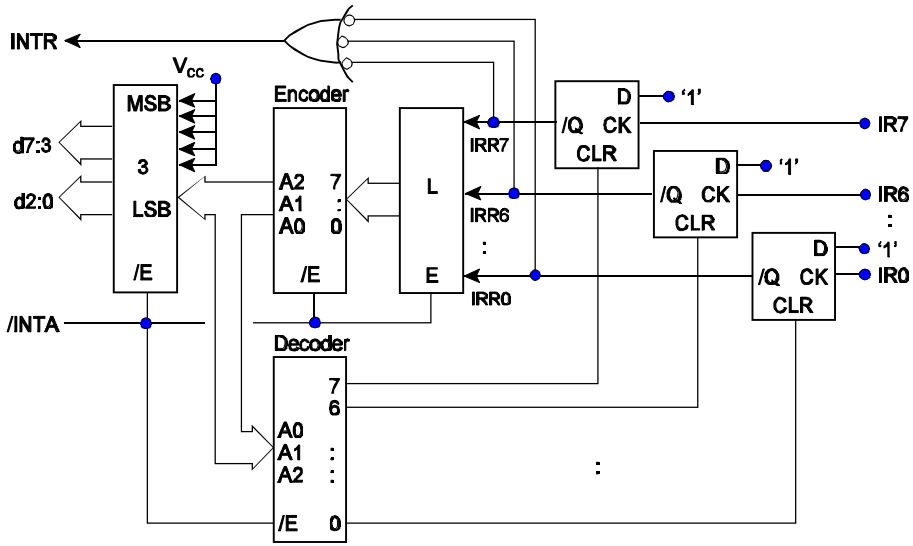
- Input 0 (active low) has highest priority.
- Input 7 (active low) has lowest priority.

/E	7	6	5	4	3	2	1	0	GS	EO	A2	A1	A0
1	x	x	x	x	x	x	x	x	1	1	1	1	1
0	1	1	1	1	1	1	1	1	1	0	1	1	1
0	x	x	x	x	x	x	x	0	0	1	0	0	0
0	x	x	x	x	x	x	0	1	0	1	0	0	1
0	x	x	x	x	x	0	1	1	0	1	0	1	0
0	x	x	x	x	0	1	1	1	0	1	0	1	1
0	x	x	x	0	1	1	1	1	0	1	1	0	0
0	x	x	0	1	1	1	1	1	0	1	1	0	1
0	x	0	1	1	1	1	1	1	0	1	1	1	0
0	0	1	1	1	1	1	1	1	0	1	1	1	1

PRIORITIZATION USING INTERRUPT CONTROLLERS

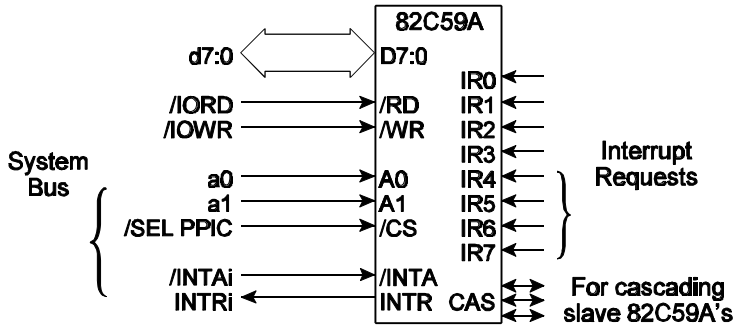
- An interrupt controller resolves priority in hardware using a priority encoder.
- The ***simple*** interrupt controller (following) implements fixed priority in H/W using an 8 to 3 priority encoder.
- **Priority is resolved at the instant /INTA is activated.**
- Operation
  - One or more active IRR's causes, INTR ← '1',
  - During the INA bus cylcle, /INTA ← '0', causing:
    - IRR state to be held at the latch,
    - A unique interrupt type number is gated onto the data bus associated with the highest priority unmasked IRRx bit (at the instant /INTA ← '0'),
    - The decoder is activated, clearing the interrupt request flip flop with highest priority, (at the instant /INTA ← '0').

A SIMPLE INTERRUPT CONTROLLER



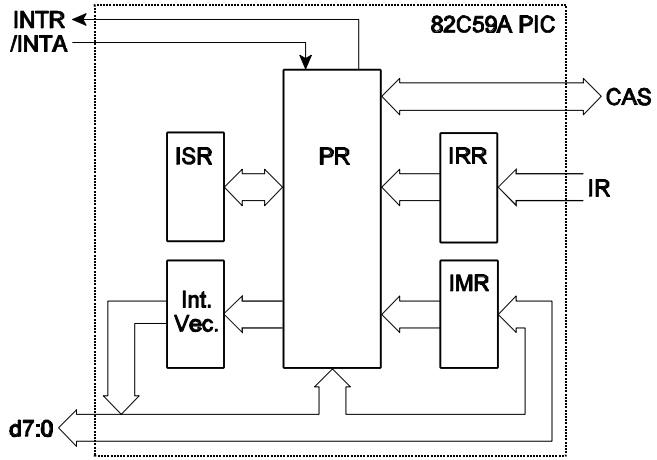
THE 82C59A PIC H/W INTERFACE

- Resolves priority at the instant /INTA ← '0',
- Also takes account of the priority of interrupt handlers that are executing!
- May cascade 82C59A's to increase number of interrupts.



THE 82C59A PIC INTERNAL COMPONENTS

- ***IRReg (Interrupt Request Register)*** : separately clearable flip flops that keep track of all pending interrupt requests at IRx. Read Only.
- ***IMReg (Interrupt Mask Register)*** : selectively masks / unmasks certain interrupt requests, IRx. (1-masks, 0-unmasks). Read / Write.
- ***ISReg (In Service Register)*** : separately clearable flip flops that keeps track of interrupt handlers that have started but have not yet finished.
- An ***end of interrupt command***, EOI, is issued within the interrupt handler to clear the associated bit in the ISReg!
- ***PR (Priority Resolver)*** : determines when INTR is activated and what TYPE number is put on the DATA pins in response to /INTA.



OPERATION OF THE 82C59A PRIORITY RESOLVER

- The PR determines which (unmasked) interrupt request, IRRx, has highest priority, if any.
- The default priority : IR0 highest, IR7 lowest.
- The PR checks the ISReg to make sure that an equal or higher priority interrupt handler (0-highest) is not executing.
- If an equal or higher priority interrupt handler ***is*** executing the PR ***waits*** until the higher priority ISReg bits are cleared before activating INTR.
- The PR also releases the correct TYPE number onto the data bus in response to /INTA.

PROGRAMMING THE 82C59A PIC'S (OF THE 82C206)

- Refer to the 82C206 data sheets in your Lab Manual!
- Consider the operation of two cascaded 82C59A's : one ***master*** and one ***slave***.
- Each 82C59A has four Initialization Control Words, ICW1, ICW2, ICW3 and ICW4 to initialize device operation.
- Each 82C59A has three Operation Control Words, OCW1, OCW2 and OCW3, that are R/W during operation.

1. Initialization Sequence

- Write ICW1 (to 0x20 Master, 0xA0 Slave) to select : triggering, cascade operation and/or start initialize.
- Write ICW2 (to 0x21 Master, 0xA1 Slave) to program the 5-MSB's of the interrupt type supplied by the PR.
- Write ICW3 = 0x04 (to 0x21 Master) indicating a slave 82C59A is connected to IR2.
- Write ICW3 = 0x02 (to 0xA1 Slave) indicating the slave 82C59A is connected to a master at IR2.
- Write ICW4 (to 0x21 Master, 0xA1 Slave).
- Write OCW1 (to 0x21 Master, 0xA1 Slave) to unmask/mask interrupt requests, (0-unmasks).

ICW1	x	x	x	SI	LT	x	S/M	x

- SI : Start initialization, fixed priority (IR7 highest)
- LT : Level/Edge triggering at IR inputs.
- S/M : Single/Multiple 82C59A's (0 - cascade operation).

ICW2	T7	T6	T5	T4	T3	x	x	x

- T7:3 Interrupt type number, 5-MSB's.

2. Controlling the 82C59A during Operation

- Write OCW1 (to 0x21 Master, 0xA1 Slave) to unmask/mask interrupt requests, (1-masks).
- Write OCW2 (to 0x20 Master, 0xA0 Slave) to tell the 82C59A that a certain ISRoutine is done, specific EOI.
- Write OCW3 (to 0x20 Master, 0xA0 Slave) before reading the ISReg or IRReg status.

3. Reading the 82C59A Status

- Read OCW1 for the IMReg status.
- Read OCW3 for the status of ISReg or IRReg, (OCW3 must first be written to select ISReg or IRReg).

PROGRAMMING THE 82C59A PIC (CONT'D)

OCW2	R = R2 R1 R0	0	0	L = L2 L1 L0

- R = 001 : Non-specific EOI.
- R = 011 : Specific EOI *************use this EOI with 'L' field in lab****
- R = 100 : Rotate on auto EOI enable.
- R = 101 : Rotate on non-specific EOI.
- R = 110 : Specific rotation.
- R = 111 : Rotate on specific EOI.
- L = interrupt level to be acted on.

OCW3	0	ESMM	SMM	x	x	PM	RR	RIS

- ESMM : Enable special mask mode.
- SMM : Special mask mode.
- PM : Polled mode.
- RR : Read IRReg.
- RIS : Read ISReg.

S/W FOR INTERRUPT DRIVEN APPLICATIONS

- ***The main program*** :

  - Initialize stack, (automatically done in “C”),
  - Initialize vector table,
  - Initialize interrupt controller(s), FLAG = reset,
  - Enable interrupt generation,
  - Do other things ...
  - Wait for FLAG == set before re-enabling interrupts from the source again.

- ***Interrupt handler*** :

  - Save state, (automatically done in “C”),
  - Read/write the first/next byte or word to IO Device,
  - If all bytes/words have been transferred then disable interrupt generation at the source and set FLAG,
  - Send end of interrupt command(s) to PIC(s),
  - Restore state, (automatically done in “C”),
  - IRET, (automatically done in “C”).

EXAMPLE

- Refer to Experiment 6.5 and the extensions of Experiment 6.5.

- Develop software that reads a 3-byte packet from a serial mouse connected to channel 1 of the TL16C452 as shown on the following page.

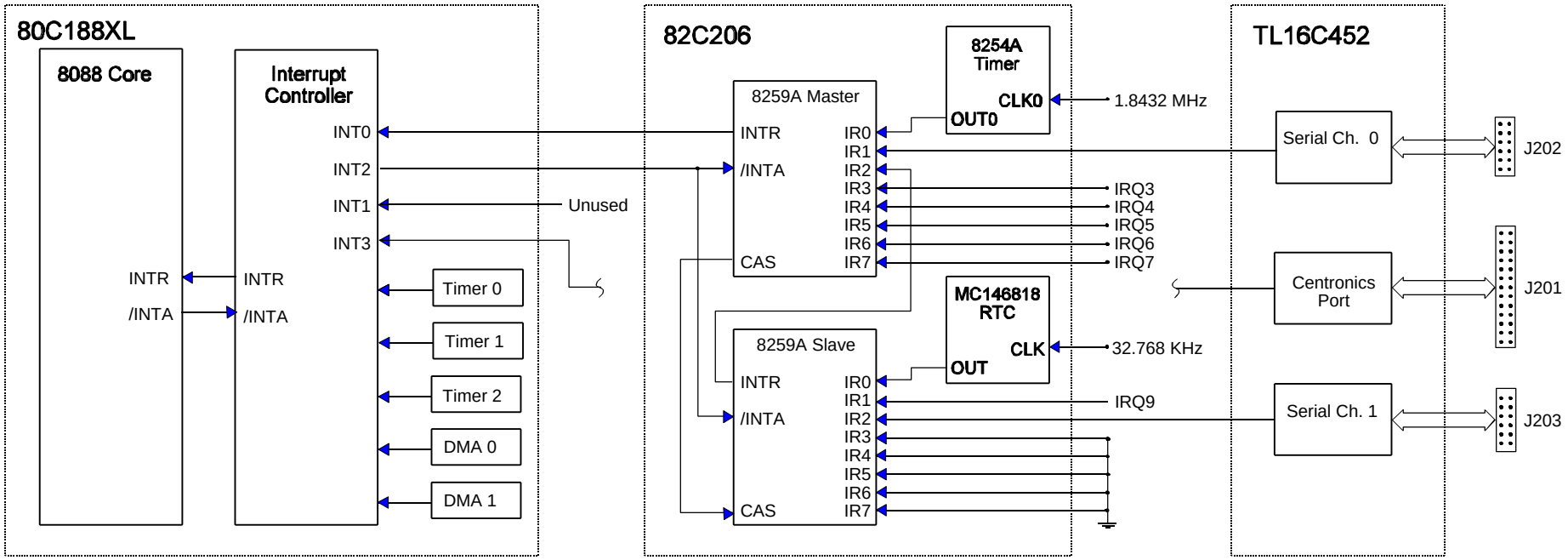
- The transfers are to take place using interrupt driven IO.

- The mouse is allowed to cause interrupts until 50 packets have been received then the serial mouse interrupt is to be masked off.

- The program should initialize the interrupt controllers and unmask the serial mouse interrupt.

- With each interrupt request a single character is transferred to the CPU via the serial channel-1 UART.

APPLICATION : THE SBC188 INTERRUPT CONTROLLER HARDWARE



IRQ3 - IRQ9 are available
at the PC/104 connector.

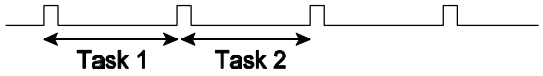
EXAMPLE (CONT'D)

```
struct packet {
      char X;
      char Y;
      char LB;
      char RB;
      };

struct packet P;          // Global for mouse packet.
char N, M, flag, T[3];    // Globals for synchronization and
                          // interrupt handler operation.
//-------------------------------------------------------------------------------
void main(void)
{
disable();                // Interrupts are masked off during
N = 0;                    // initialization.
M = 0;
P.X = 0;
P.Y = 0;
P.RB = 0;
P.LB = 0;

int_vect_init();          // Initialize interrupt vector table.
enable();                 // Unmask interrupts = set IF.
int_con_init();           // Initialize interrupt controllers.
uart_int_init();          // Initialize Data Ready interrupt in
                          // the 16C452 UART 1 receiver.

// Main program continuously displays mouse packet, P.
while(TRUE) {
      MOUSE_DISPLAY(P);
      }
}
//-------------------------------------------------------------------------------
void interrupt far mouse_int_handler(void)
{
char x;
x = inportb(RBR);         // Read character at the RBR.
if (M <= 50) {

// N keeps track of which character, (1-st, 2-nd or 3-rd) of the
// mouse packet has just been  read.
      switch(N) {
            case 0:  if((x & 0x40) != 0) {
                  T[0] = x;
                  N = 1;
                  }
                  break;
            case 1: T[1] = x;
                  N = 2;
                  break;
            case 2: T[2] = x;   // Update P if the last char of the
                  N = 0;        // packet has been received
                  Update P.X, P.Y, P.RB and P.LB
                  M = M + 1;    // M = # of packets received.
            }
      }
else {                        // Turn mouse interrupts off.
      Mask off interrupts at IR2 of the slave 8259A
      Mask off interrupts at IR2 of the master 8259A
      Mask off interrupts at INT0 of the 80C188XL
      }
Send specific End of Interrupt, EOI, to the master for IR2.
Send specific EOI to the slave for IR2.
Send an EOI to 80C188XL interrupt controller for INT0.
}
//-------------------------------------------------------------------------------
void int_vect_init(void)
{
asm {
      push ds
      push bx
      push ax

      mov ax, 0
      mov ds, ax

      mov bx, 4*(SLAV_TYPE + 2)
      mov word ptr [bx], offset mouse_int_handler
      mov word ptr [bx+2], seg mouse_int_handler

      pop ax
      pop bx
      pop ds
      }
}
//-------------------------------------------------------------------------------
void int_con_init(void)
{
outportb(ICW1_MAST, 0x18);        // Level triggered, cascaded,
outportb(ICW1_SLAV, 0x18);        // master - slave 8259A's.
```

```
// Program interrupt base types: 0xF0 for master, 0xF8 for slave.
outportb(ICW2_MAST, MAST_TYPE);
outportb(ICW2_SLAV, SLAV_TYPE);

outportb(ICW3_MAST, 0x04);      // Slave is connected to IR2.
outportb(ICW3_SLAV, 0x02);      // Slave controller address.

// Program master and slave for single, and normal EOI.
outportb(ICW4_MAST, 0x00);
outportb(ICW4_SLAV, 0x00);

outportb(OCW1_MAST, 0xFF);      // Mask off master IR0 - IR7.
outportb(OCW1_SLAV, 0xFF);          //  Mask off slave IR0 - IR7.

// Program the 80C188 interrupt controller for INT0 with high level
// trigger, cascade, not special function nested, priority 000.
outport(I0CON, 0x0030);

outportb(OCW1_MAST, 0xFB);      // Unmask IR2 in master.
outportb(OCW1_SLAV, 0xFB);          // Unmask IR2 in slave.
}
```

CONSIDERATIONS FOR USING INTERRUPTS

- A ***task*** is like a program dedicated for a specific function.
- If many tasks are in the process of being executed by a microcomputer, they are said to execute ***concurrently*.**
- The microcomputer is said to support ***multi-tasking***.
- If only one microprocessor is used then only one task can be executed at a time.
- If more than one microprocessor is used the system is referred to as a ***multiprocessor*** system.
- Each task may be allocated a maximum CPU processing time, sometimes called a ***time slice***.



- The time slice is synchronized to a ***real time clock*** interrupt.
- After a time slice, the execution of the present task is ***suspended***, its state is saved (so that execution may be resumed later) and control transfers to the next task.

REENTRANT PROCEDURES

- If a procedure being executed is interrupted and the same procedure is called in the interrupt handler then the subroutine is said to be ***reentered***.
- If reentrant procedures are used then the programmer must ensure that there is no data loss,
- i.e., all parameters must be passed in registers or on the stack and all intermediate results must be computed in registers or on the stack.

CRITICAL SECTIONS

- If certain code cannot be interrupted by another task then this code is said to be a ***critical section***.
- Interrupts may need to be disabled during critical sections.
- A significant amount of work has been done in the field of ***Operating Systems*** where interrupts need not be disabled during critical sections.
- Special software structures such as ***semaphores*** are commonly used.